VCL Mini Project - 3

Project Report: Road Image Segmentation

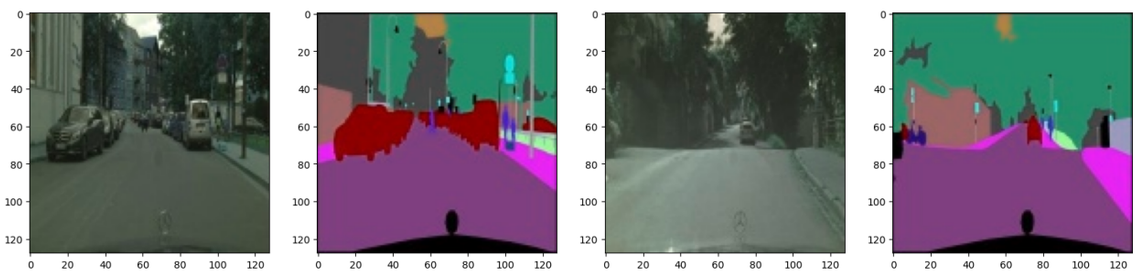
Problem Statement

The objective of this project is to design a convolutional neural network for performing segmentation on road images using the given dataset.

Semantic segmentation is a computer vision task that involves classifying each pixel in an image into specific semantic categories, enabling precise object localization and recognition within the scene. It provides a detailed understanding of the visual content, making it a crucial step in applications like autonomous driving and scene understanding.

Dataset Description

The dataset used for this project is Cityscapes 128. It contains images of urban scenes captured from a car driving through a city. Each image is labelled with pixel-level annotations indicating the different semantic classes, including roads, buildings, pedestrians, vehicles, etc.



Model Description

For the initial implementation, a U-Net architecture was trained on the Cityscapes 128 dataset. Below is the PyTorch model summary:

It comprises several key components:

1. **Encoder**: The network begins with an encoder portion consisting of multiple convolutional layers. Each convolutional layer is followed by ReLU activation and Batch Normalization. These layers are responsible for extracting high-level features from the input image.
2. **Max Pooling**: After each set of convolutional layers, a max pooling operation is applied to downsample the spatial dimensions. This helps in reducing the computational complexity while preserving important features.
3. **Decoder**: The decoder section consists of ConvTranspose2d layers, transposed convolutions or deconvolutions. These layers perform the inverse pooling operation, allowing the network to upsample and recover spatial information.
4. **Skip Connections**: SegNet also employs skip connections, which bypass one or more layers. This enables the network to retain fine-grained details from earlier layers during the upsampling process.
5. **Final Convolutional Layer**: The model concludes with a final convolutional layer that outputs the segmented image. In this case, it produces 12 channels, corresponding to the 12 classes in the semantic segmentation task.

The total number of parameters in the model is approximately 18 million, making it a relatively complex network. This complexity is essential for capturing intricate details and nuances in road images.

For training, the Adam optimizer was used. Adam is an adaptive learning rate optimization algorithm that adjusts the learning rates of individual parameters, allowing for faster convergence and better performance.

Categorical Cross Entropy loss is apt for semantic segmentation tasks where each pixel is assigned to a specific class. It computes the cross entropy between the predicted class probabilities and the true class labels. This loss function is well-suited for training the network to accurately classify pixels into their respective semantic categories.

Overall, combining the SegNet architecture, Adam optimizer, and Categorical Cross Entropy loss function forms a robust framework for training the model to perform semantic segmentation on road images.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
DataParallel                             [1, 12, 128, 128]         --
├─SegNet: 1-1                            [1, 12, 128, 128]         --
│    └─Sequential: 2-1                   [1, 64, 128, 128]         --
│    │    └─Conv2d: 3-1                  [1, 64, 128, 128]         1,792
│    │    └─ReLU: 3-2                    [1, 64, 128, 128]         --
│    │    └─BatchNorm2d: 3-3             [1, 64, 128, 128]         128
│    │    └─Conv2d: 3-4                  [1, 64, 128, 128]         36,928
│    │    └─ReLU: 3-5                    [1, 64, 128, 128]         --
│    │    └─BatchNorm2d: 3-6             [1, 64, 128, 128]         128
│    └─MaxPool2d: 2-2                    [1, 64, 64, 64]           --
│    └─Sequential: 2-3                   [1, 128, 64, 64]          --
│    │    └─Conv2d: 3-7                  [1, 128, 64, 64]          73,856
│    │    └─ReLU: 3-8                    [1, 128, 64, 64]          --
│    │    └─BatchNorm2d: 3-9             [1, 128, 64, 64]          256
│    │    └─Conv2d: 3-10                 [1, 128, 64, 64]          147,584
│    │    └─ReLU: 3-11                   [1, 128, 64, 64]          --
│    │    └─BatchNorm2d: 3-12            [1, 128, 64, 64]          256
│    └─MaxPool2d: 2-4                    [1, 128, 32, 32]          --
│    └─Sequential: 2-5                   [1, 256, 32, 32]          --
│    │    └─Conv2d: 3-13                 [1, 256, 32, 32]          295,168
│    │    └─ReLU: 3-14                   [1, 256, 32, 32]          --
│    │    └─BatchNorm2d: 3-15            [1, 256, 32, 32]          512
│    │    └─Conv2d: 3-16                 [1, 256, 32, 32]          590,080
│    │    └─ReLU: 3-17                   [1, 256, 32, 32]          --
│    │    └─BatchNorm2d: 3-18            [1, 256, 32, 32]          512
│    └─MaxPool2d: 2-6                    [1, 256, 16, 16]          --
│    └─Sequential: 2-7                   [1, 1024, 16, 16]         --
│    │    └─Conv2d: 3-19                 [1, 1024, 16, 16]         2,360,320
│    │    └─ReLU: 3-20                   [1, 1024, 16, 16]         --
│    │    └─BatchNorm2d: 3-21            [1, 1024, 16, 16]         2,048
│    │    └─Conv2d: 3-22                 [1, 1024, 16, 16]         9,438,208
│    │    └─ReLU: 3-23                   [1, 1024, 16, 16]         --
│    │    └─BatchNorm2d: 3-24            [1, 1024, 16, 16]         2,048
│    └─ConvTranspose2d: 2-8              [1, 256, 32, 32]          2,359,552
│    └─Sequential: 2-9                   [1, 256, 32, 32]          --
│    │    └─Conv2d: 3-25                 [1, 256, 32, 32]          1,179,904
│    │    └─ReLU: 3-26                   [1, 256, 32, 32]          --
│    │    └─BatchNorm2d: 3-27            [1, 256, 32, 32]          512
│    │    └─Conv2d: 3-28                 [1, 256, 32, 32]          590,080
│    │    └─ReLU: 3-29                   [1, 256, 32, 32]          --
│    │    └─BatchNorm2d: 3-30            [1, 256, 32, 32]          512
│    └─ConvTranspose2d: 2-10             [1, 128, 64, 64]          295,040
│    └─Sequential: 2-11                  [1, 128, 64, 64]          --
│    │    └─Conv2d: 3-31                 [1, 128, 64, 64]          295,040
│    │    └─ReLU: 3-32                   [1, 128, 64, 64]          --
│    │    └─BatchNorm2d: 3-33            [1, 128, 64, 64]          256
│    │    └─Conv2d: 3-34                 [1, 128, 64, 64]          147,584
│    │    └─ReLU: 3-35                   [1, 128, 64, 64]          --
│    │    └─BatchNorm2d: 3-36            [1, 128, 64, 64]          256
│    └─ConvTranspose2d: 2-12             [1, 64, 128, 128]         73,792
│    └─Sequential: 2-13                  [1, 64, 128, 128]         --
│    │    └─Conv2d: 3-37                 [1, 64, 128, 128]         73,792
│    │    └─ReLU: 3-38                   [1, 64, 128, 128]         --
│    │    └─BatchNorm2d: 3-39            [1, 64, 128, 128]         128
│    │    └─Conv2d: 3-40                 [1, 64, 128, 128]         36,928
│    │    └─ReLU: 3-41                   [1, 64, 128, 128]         --
│    │    └─BatchNorm2d: 3-42            [1, 64, 128, 128]         128
│    └─Conv2d: 2-14                      [1, 12, 128, 128]         6,924
==========================================================================
Total params: 18,010,252
Trainable params: 18,010,252
Non-trainable params: 0
Total mult-adds (G): 15.85
==========================================================================
Input size (MB): 0.20
Forward/backward pass size (MB): 142.08
Params size (MB): 72.04
Estimated Total Size (MB): 214.32
==========================================================================
```

Techniques Incorporated

To improve the performance of the model, a combination of two losses was incorporated:

1. Multiclass Dice Loss

Dice loss is a metric used for binary segmentation tasks. However, for multiclass segmentation, it can be extended to handle multiple classes. The Dice loss for a single class i is defined as:

$Dice_i = \frac{2 \cdot \text{TP}_i}{2 \cdot \text{TP}_i + \text{FP}_i + \text{FN}_i}$

Where:

- TP_i is the true positive count for class i,
- FP_i is the false positive count for class i,
- FN_i is the false negative count for class i.

The overall Dice loss for multiclass segmentation is the average of the Dice losses for all classes.

The performance on Multiclass Dice Loss alone was very insignificant. Hence, I employed it in combination with Categorical Cross Entropy.

The weight of both the losses with each other was empirically found to be 1:1.

2. Categorical Cross Entropy Loss

Categorical Cross Entropy loss is a standard function for multiclass classification tasks. It measures the dissimilarity between the predicted class probabilities and the true class labels.

The Categorical Cross Entropy loss for a single example is defined as:

$\text{CE}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$

Where:

- N is the number of classes,
- y is the true class distribution (one-hot encoded),
- $\hat{y}$ is the predicted class probabilities.

The overall loss is the average over all examples in the batch.

Dataset Augmentation

To make the model more robust and capable of handling variations in real-world scenarios, the dataset was augmented with the following techniques:

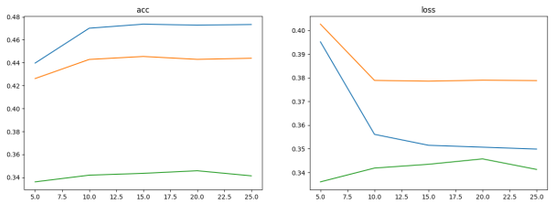
1. Random Rotations: Images were randomly rotated by $\pm 25^\circ$ to simulate different road orientations.
2. Gaussian Noise: Gaussian noise was added to the images to mimic noise in real-world images.

Testing

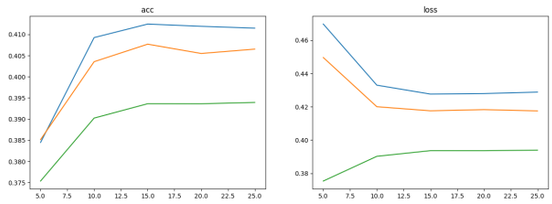
The metrics retrieved from training the Model on both Original and Augmented Data are juxtaposed below.

Epochs\Score	Training IOU	Training Loss	Validation IOU	Validation Loss	Validation IOU (Augmented)	Validation Loss (Augmented)
5	43.97	0.3952	42.62	0.4026	33.61	0.6399
10	47.01	0.3561	44.29	0.3749	34.19	0.6711
15	47.36	0.3515	44.54	0.3788	34.35	0.6649
20	47.28	0.3507	44.29	0.3790	34.37	0.6693
25	47.33	0.3499	44.39	0.3788	34.13	0.6707

Epochs\Score	Training IOU	Training Loss	Validation IOU	Validation Loss	Validation IOU (Augmented)	Validation Loss (Augmented)
5	38.45	0.4697	38.52	0.4496	37.54	0.4754
10	40.92	0.4276	40.35	0.4200	39.02	0.4517
15	41.24	0.4280	40.77	0.4176	39.36	0.4512
20	41.19	0.4288	40.55	0.4183	39.36	0.4491
25	41.15	0.4280	40.65	0.4175	39.39	0.4481

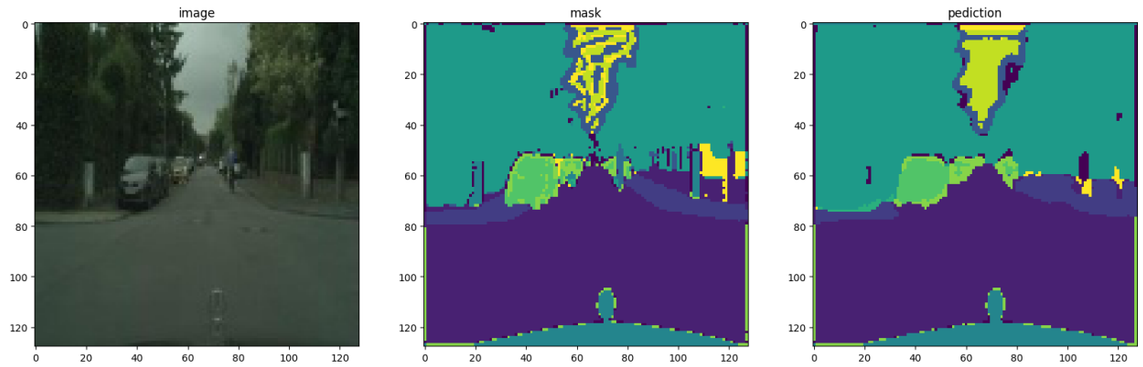


Model Training on Original Data (🔵 Training, 🔴 Testing (Original Split), 🟢 Testing (Augmented Split))

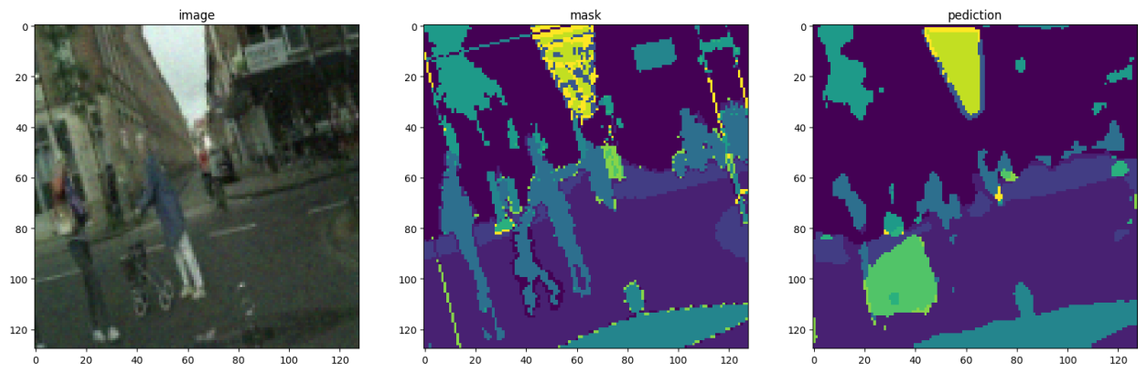


Model Trained with Augmented + Original Data (🔵 Training, 🔴 Testing (Original Split), 🟢 Testing (Augmented Split))

Visualization of Results



Original Training Data



Augmented Training Data

Codebase

https://github.com/eternal-f1ame/MP-3-VCL

Deployed App

https://semseg.streamlit.app/

Snapshots